Demystifying Performance Predictions of Distributed FFT3D Implementations

Daniel Orozco, Elkin Garcia, Robert Pavel, Orlando Ayala, Lian-Ping Wang, and Guang Gao

University of Delaware
{orozco@, egarcia@, rspavel@, omayalah@, lwang@, ggao@capsl. }udel.edu

Abstract. This paper presents a comprehensive story of the development of simpler performance models for distributed implementations of the Fast Fourier Transform in 3 Dimensions (FFT3D). We start by providing an overview of several implementations and their performance models. Then, we present arguments to support the use of a simple power function instead of the full performance models proposed by other publications. We argue that our model can be obtained for a particular problem size with minimal experimentation while other models require significant tuning to determine their constants.

Our advocacy for simpler performance models is inspired by the difficulties found when estimating the performance of FFT3D programs. Correctly estimating how well large-scale programs (such as FFT3D) will work is one of the most challenging problems faced by scientists. The significant effort devoted to this problem has resulted in the appearance of numerous works on performance modeling.

The results produced by an exhaustive performance modeling study may predict the performance of a program with a reasonably good accuracy. However, those studies may become unusable because their aim for accuracy can make them so difficult and cumbersome to use that direct experimentation with the program may be preferable, defeating their original purpose.

We propose an alternative approach in this paper that does not require a full, accurate, performance model. Our approach mitigates the problem of existing performance models, each one of the parameters and constants in the model has to be carefully measured and tuned, a process that is intrinsically harder than direct experimentation with the program at hand.

Instead, we were able to simplify our approach by (1) building performance models that target particular applications in their normal operating conditions and (2) using simpler models that still produce good approximations for the particular case of a program's normal operating environment.

We have conducted experiments using the Bluefire Supercomputer at the National Center for Atmospheric Research (NCAR), showing that our simplified model can predict the performance of a particular implementation with a high degree of accuracy and very little effort when the program is used in its intended operating range.

Finally, although our performance model does not cover extreme cases, we show that its simple approximation under the normal operating conditions of FFT3D is able to provide solid, useful approximations.

1 Introduction

The Fast Fourier Transform (FFT)[1] is an important tool in physics and engineering, and it is relevant to numerous High Performance Computing (HPC) applications.

This paper addresses the issue of how to efficiently select an FFT algorithm for the particular case of large-scale distributed scientific simulations. Examples of large-scale scientific applications that use FFT include Cloud Physics[2], Molecular Dynamics[3], calculation of Coulomb Energies [4], Seismic Imaging [5] and Computational Geosciences [6]

Understanding the performance of FFT in large scale systems is important because FFT plays a significant role in many scientific programs. For example, the turbulent droplet collision simulation developed by Wang et al[7] uses a pseudo-spectral method for the flow simulation that requires multiple 3-Dimensional FFTs during each timestep of the computation. In fact, profiling of the flow simulation alone reveals that the 3D FFT alone can take up to 90% of the total execution time in the program.

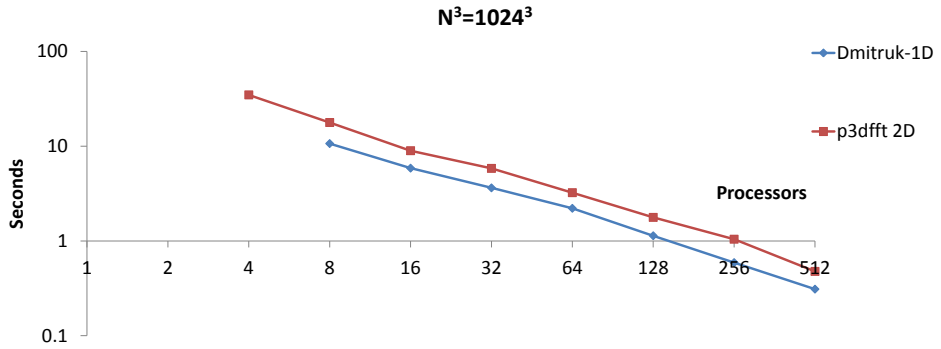

Fig. 1. Performance of two different FFT3D techniques. The relationship between time to execute and the number of processors appears to follow a straight line in this logarithmic plot.

Because the FFT is not the objective of the computation in many cases, but rather a tool used to help it, it is of paramount importance to understand the effect of design choices on the overall program. For example, there may be a certain overhead in the passing of arguments to the FFT. The partitioning of the data may force the use of certain FFT strategies, or even the physical location of the compute nodes may greatly affect the performance of particular FFT strategies.

The necessity to understand the performance of FFT implementations and its relationship to the program using it has prompted researchers to build performance models that aim to predict the performance of particular implementations. These previous approaches [8–11] have resulted in very detailed performance models that include many factors such as computation time, communication time, the presence of overlapping or pipelining, the location of the memory and so on.

The biggest drawback to building a performance model is that correctly developing each performance model requires significant effort: the algorithm must be considered, constants in the model have to be measured, and the model may only work for certain architectures or interconnects. The difficulties of building performance models are so great that when one is correctly found it is published in a recognized venue (*e.g.* Dimitruk published his model in the *Parallel Computing* Journal [8] and Ayala's model[11] is currently under review in the same journal).

Instead, we have looked at alternatives that would allow scientists to make decisions about 3-Dimensional FFT implementations without the burden of developing full performance models. We have approached the problem by studying the nature of the 3D-FFT computation itself and existing performance models for different 3D-FFT strategies to see if there is a simpler way to efficiently select a particular 3D-FFT implementation over a set of alternatives.

We have sought to understand the performance of different 3D-FFT implementations on distributed memory machines through preliminary experiments (Figure 1) conducted on the Bluefire Supercomputer from the National Center for Atmospheric Research (NCAR).

Figure 1 shows the results of using two very different implementations of FFT3D on data of the same size. The differences in performance between the two techniques shown are due to differences in the implementation such as communication strategy and algorithm used.

Our experiments, which *focus on large-scale systems*, suggest an alternative approach to analyze FFT3D implementations: The running time of a particular FFT3D implementation can be approximated by a line in a logarithmic graph. This approximation corresponds to a power function. The power-function approach has the advantage of being significantly easier to use and build while still allowing the estimation (albeit with reduced accuracy when communication latency is high) of the performance of particular 3D-FFT implementations.

To test our hypothesis regarding the validity of our power-law approximation, we have assembled several 3D-FFT implementations. A brief overview for each one of them is given in Section 2, along with their performance models, when they are available. We develop our prediction model in Section 3 and we conduct

N^3: Problem size	P: Number of processors
B_w: Inter-process bandwidth	k_c: Average computation rate (FLOPS)
k_m: A constant associated with main memory bandwidth	T: Predicted execution time
k_s: A constant associated with the startup time for a message	

Table 1. Parameter and constant definitions

experiments to test the validity of our model in Section 4. Finally, we provide some conclusions in Section 5.

2 Background on FFT3D Algorithms

Many techniques have been investigated and developed to perform a distributed 3-dimensional FFT. For the purpose of this study, we focused on approaches that did not change the underlying FFT algorithm but instead changed the decomposition of the data and the communication patterns. The algorithms presented here leverage the principle that a 3-dimensional FFT can be performed by executing a 1-dimensional FFT along each one of the three dimensions of the data.

We will use the notation in Table 1 to aid us in our explanation of the FFT3D algorithms and models.

2.1 1D Domain Decomposition: "Slabs"

The first decomposition we studied is the 1D domain decomposition, as seen in Figure 2. This decomposition is often referred to as a "slab" decomposition where each processor is responsible for a single slab. In the case of Figure 2, each processor initially holds a slab of size $N \times N \times (N/P)$.

Because a slab must have a width of at least one element, the parallelism of 1D decomposition implementations is limited to $P \leq N$ processors.

Dmitruk's 1D Domain Decomposition Dmitruk's solution to the problem of the 3-Dimensional FFT [8] is to partition data in one of the space dimensions.

Each of the P processors is responsible for a single slab. First, each processor performs N/P 2D FFTs along the xy plane. Then, a transpose is performed such that the data in the z direction is now saved in the y direction and vice versa. With this transposed data, the required 1D FFTs are performed. Finally, a second transpose may be performed to restore the data to its original orientation, depending upon the needs of the application.

However, the essence of Dmitruk's work is in his implementation of the transpose. There are three methods that work on the principle of breaking the slab into "blocks", moving the blocks to the appropriate processors, and then locally rearranging the data. What differs is the communication scheme used. The first

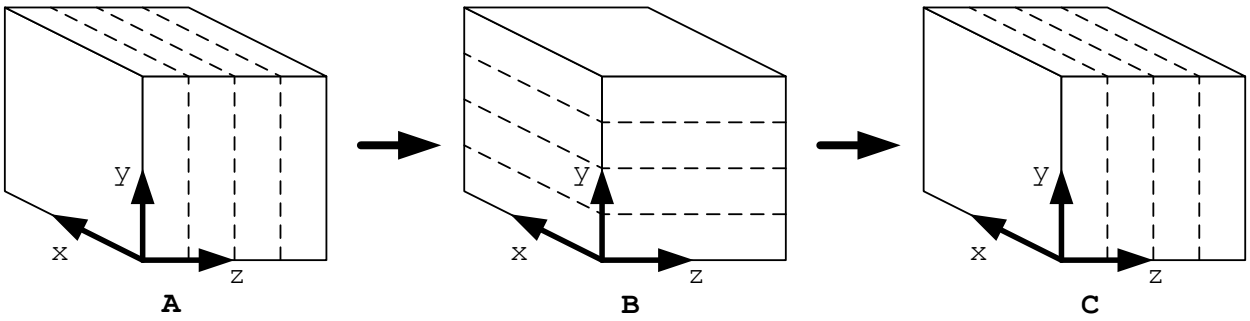

Fig. 2. 1D domain (slab) decomposition for FFT 3D

method involves a cyclic strategy: Each processor sends data to the next processor in the list while receiving data from the previous processor. This is repeated until every processor has traversed the entire list of processors. The second strategy is similar in that each processor sends to the processor that is currently receiving from via a temporary pairing strategy. This is repeated in parallel until every processor has paired with every other processor. Finally, MPI collectives in the form of the "all-to-all" operator were used for the third method. Of these three methods, the second method's pairwise strategy gave the best performance under Dmitruk's tests.

Equation 1 shows the full performance model presented by Dmitruk's paper[8].

$$T = \frac{2N^3}{B_w P} + \frac{15}{2\log(2)} \frac{N^3 \log(N)}{k_c P} + \frac{3N^3}{k_m P} + 2k_s P \tag{1}$$

2.2 2D Domain Decomposition: "Pencils"

A 2D domain decomposition strategy is a natural extension to the 1D idea. The 2D domain decomposition can be done along X axis, referred to as X-pencil, Y axis (Y-pencil) or Z axis (Z-pencil).

Assuming that the P processors are arranged in a grid so that $P = P_y \times P_z$, Figure 2 shows a case where each pencil is of size $N \times (N/P_y) \times (N/P_z)$ and each processor is responsible for a single pencil.

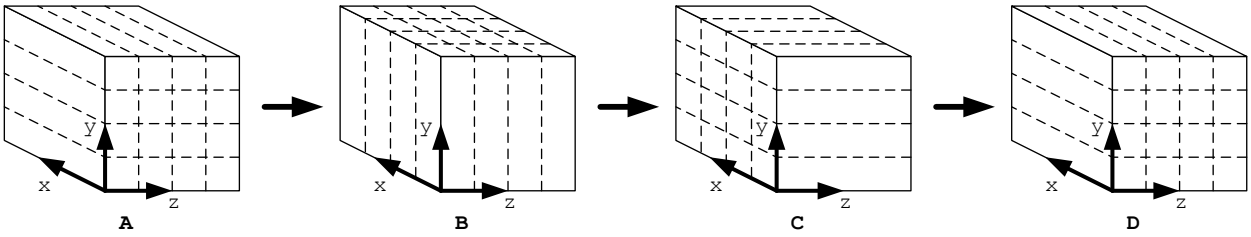

Fig. 3. 2D domain (pencil) decomposition for FFT 3D

The three libraries studied below follow the same general pattern. Three sets of 1D FFTs are performed along each axis with a transpose used to reorder data. In the case of Figure 3, this means 1D FFT in the x-direction, transpose, y-direction, transpose, and then z-direction. Depending on the application, a final transpose will be performed to restore the data. These libraries differ in how the transpose is handled.

2DECOMPFFT 2DECOMPFFT [9] is designed for applications using 3-dimensional structured mesh and spatially implicit numerical algorithms. It implements a general-purpose 2D pencil decomposition for data distribution on distributed memory platforms. The transpositions are done using the MPI "all-to-all" operator, but with a complex communication pattern according to the orientation of pencils and their associated memory pattern.

p3dfft The p3dfft algorithm [10] is an algorithm developed by Pekurovsky that uses a 2D pencil decomposition with the objective of maximizing parallelism. p3dfft was written in Fortran and MPI and uses FFTW[12] as the underlying library for FFT. Communication operations are done using MPI "all-to-all" operations.

Pekurovsky's FFT3D requires a total of 3 transpositions to compute a transformation. Data is redistributed during each step using call to MPI All-to-All that requires internal redistribution of data. In each of the 3 communication steps, each processor transfers a block of size N^3/P with a bandwidth of B_w. Equation 2 summarizes these steps into a performance model.

$$T = \frac{3N^3}{B_w P} + 3\frac{N^3 \log(N)}{k_c P} + \frac{3N^3}{k_m P} + 2k_s(P_y + P_z - 2) \tag{2}$$

Ayala's 2D Domain Decomposition Ayala and Wang [11] built upon the work of Dmitruk et al. As with the other libraries, a 2D Pencil Decomposition is used. Due to the target application of Direct Numerical Simulation, the last transpose is not necessary in Ayala's method.

As with Dmitruk's work, the optimization of the transpose plays a large part in improving performance. In this case, Ayala chose to follow Dmitruk's cyclic strategy. This provides better performance because during a transpose for a 2D Decomposition, only processors in the same plane as the transpose need to communicate. For example, a transpose along the x and y axes will only require communication between processors in the same xy plane to communicate.

Equation 3 presents a performance model taken from Ayala and Wang's paper[11].

$$T = \frac{4N^3}{B_w P} + \frac{15}{2\log(2)} \frac{N^3 \log(N)}{k_c P} + \frac{6N^3}{k_m P} + 2k_s(P_y + P_z - 2) \tag{3}$$

2.3 3D Domain Decomposition: "Blocks"

Finally, we studied the 3D Domain Decomposition in which the data is broken up into blocks. The P processors that participate in the computation are arranged in a 3-Dimensional grid so that $P = P_x \times P_y \times P_z$. The amount of data held by each processor (Figure 4) is $(N/P_x) \times (N/P_y) \times (N/P_z)$.

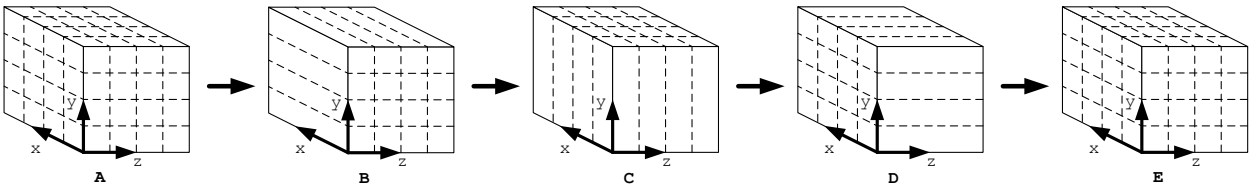

Fig. 4. 3-Dimensional Decomposition of Data. Each one of the small cubes represent the data held by each processor.

3Decomp 3Decomp computes FFT3D by executing four data transpositions as shown in Figure 4. In each case the communication is done using nonblocking point-to-point MPI calls. Due to the differences in topology between each step, each processor needs to communicate with several other processors. The local 1-Dimensional FFT transforms required are computed using the FFTW[12] library.

The performance model of 3Decomp (Equation 4) is obtained by analyzing its data movement and the amount of computation performed: Four communication steps are used where all data held by each processor (N^3/P in size) is reorganized locally and then sent to another processor. Each processor, in parallel, computes $3N^2/P$ one-dimensional FFTs, each of complexity $N\log(N)$.

$$T = \frac{4N^3}{B_w P} + 3\frac{N^3 \log(N)}{k_c P} + \frac{4N^3}{k_m P} + 2k_s(2P_x + P_y(P_x + P_z) - 4) \tag{4}$$

3 A simplified model approach for FFT3D

Section 2 presented a selection of algorithms for FFT3D and their performance models.

Correctly developing and using performance models such as those presented in Equations 1, 2, 3 and 4 require a significant effort: It requires a reasonable understanding of the implementation, and it requires extensive testing on the machine to identify the constants of the model.

Our previous experiments (Figure 1) showed a case where plotting execution time as a function of the number of processors resulted in a straight line when plotted on a logarithmic scale, indicating a power-function behavior. These results have motivated us to approximate the performance models of FFT3D as a power function as presented in Equation 5.

$$T_N(P) = A_N \times P^{-B_N} \tag{5}$$

Equation 5 is significantly simpler than other performance models, such as those presented in Section 2, and it is appropriate to represent the behavior observed in Figure 1.

The power-function approximation is intended to provide estimates of parallelism for large scale systems in a range of situations where there is enough parallelism available. *i.e.* Equation 5 will provide a reasonable approximation for a particular number of processors if the FFT3D implementation is able to scale reasonably well up to that number of processors.

It is worth noting that our approximation doesn't account for the startup latency of a message. Thus, as P approaches N, the accuracy of our model may decrease. The following section provides experiments that test our hypothesis and the usability of our simplified performance model, even under these circumstances.

4 Experiments

We conducted experiments to confirm the validity of our claims. Specifically, we wanted to test whether or not the use of the power-law model of Equation 5 was sufficient to predict the performance of particular implementations of FFT3D.

Our experiments were conducted on the Bluefire Supercomputer at the National Center for Atmospheric Research (NCAR). The Bluefire supercomputer possesses 4096 Power6 processors running at 4.7GHz and having 2GB of memory. Bluefire's processors are connected by eight 4X infiniband DDR links with a total bandwidth of 2.5GB/s. Further information about Bluefire's hardware can be obtained from NCAR's website[13].

The experiments seek to establish whether or not it is possible to predict the performance of FFT3D implementations using the model of Equation 5. As explained in Section 3, the power-law model attempts to predict the performance of a particular implementation in common situations.

To test our hypothesis, we have ran the implementations described in Section 2. In our experiments, we chose several input sizes ($N^3 = 256^3$, $N^3 = 512^3$ and $N^3 = 1024^3$) for FFT3D that are typical of large-scale programs such as in cloud microphysics studies by direct numerical simulations. For each implementation, we run each problem size while the number of processors was changed. Due to memory, execution time, or processor count constraints, not all combinations of problem sizes and processor counts were possible.

The results of our experiments are reported in Figures 5, 6 and 7. To confirm the validity of our model, we have build simplified performance models for each implementation in the style of Equation 5. For each case, the constants A_N and B_N found in Equation 5 have been found using a least squares regression model.

The resulting models are presented in Table 2. The table shows the models as well as an indication (R^2 parameter) of how well the models are able to represent the experimental data obtained ($R^2 = 1$ means perfect match).

As can be seen from Figures 5, 6 and 7 and Table 2, the power function approximation works remarkably well. In most situations, the R^2 parameter is greater than 0.99, indicating that the model was able to fit the data with a high degree of accuracy.

Figures 5, 6 and 7 also give an intuitive indication of the quality of our models: The predicted models, represented with dashed lines, agree with the data gathered (plotted with solid lines).

The high level of similarity between the solid lines (measured) and dashed lines (model) and the high value of the R^2 indicator lead us to conclude that our model is an effective way to predict the performance of FFT3D implementations in their normal operating range. This is an important conclusion because the simple, power-function model can be quickly obtained with as few as only two measurements, one at low processor counts, and one at high processor counts. This is significantly simpler than the other models presented in Section 2 while still presenting good accuracy.

Of particular interest was the issue of startup latency and congestion when the maximum parallelism of a problem is approached. This is most evident for Dmitruk's 1D Domain Decomposition when $N = P$. Figure 5 does show this limitation. However, as the problem size is increased, the impact of the startup latency is greatly decreased, and our approximation's accuracy remains high.

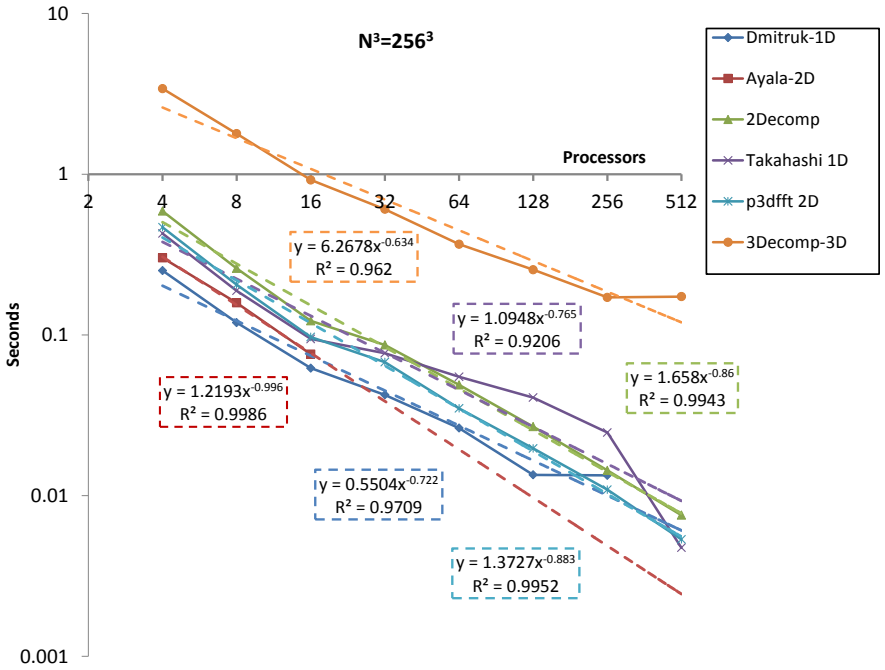

Fig. 5. FFT3D performance for $N^3 = 256^3$. Solid lines show actual experimental data. Dashed lines show a prediction using a power function.

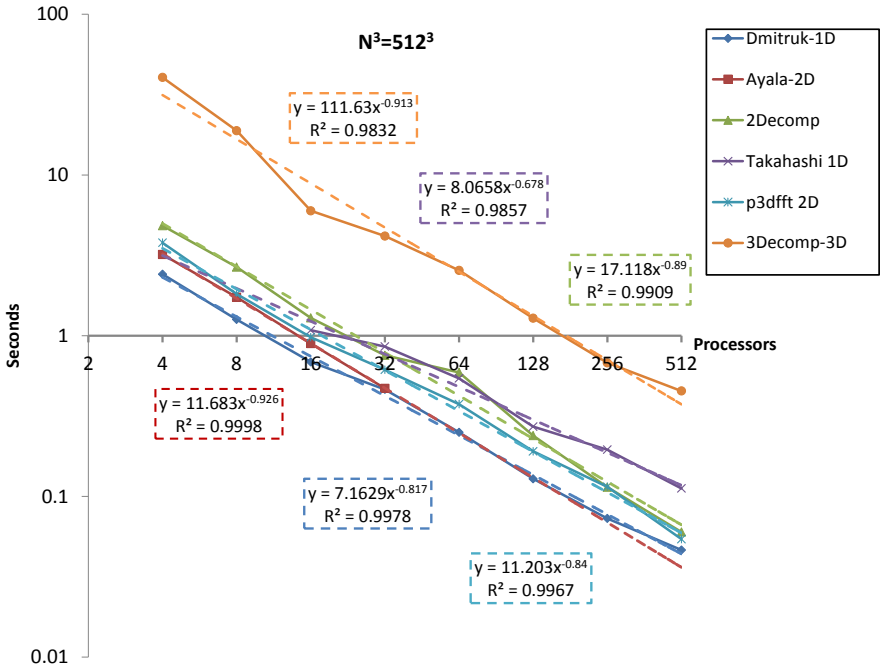

Fig. 6. FFT3D performance for $N^3 = 512^3$. Solid lines show actual experimental data. Dashed lines show a prediction using a power function.

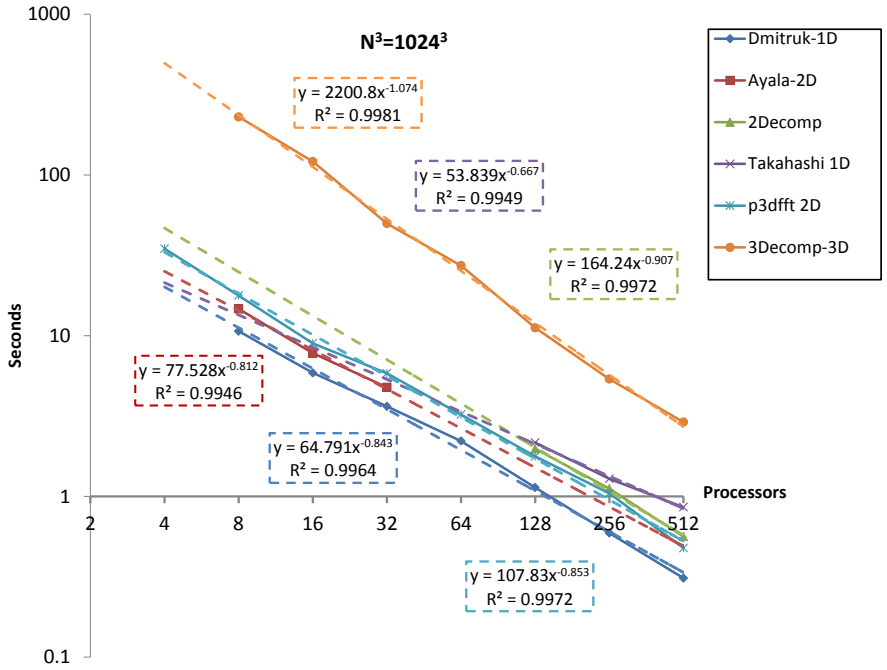

Fig. 7. FFT3D performance for $N^3 = 1024^3$. Solid lines show actual experimental data. Dashed lines show a prediction using a power function.

Models for $N^3 = 256^3$		
Implementation	**Function**	R^2
Dmitruk-1D	$T_{256} = 0.550P^{-0.722}$	0.970
Ayala-2D	$T_{256} = 1.22P^{-0.996}$	0.999
2Decomp	$T_{256} = 1.658P^{-0.86}$	0.994
Takahashi 1D	$T_{256} = 1.095P^{-0.765}$	0.921
p3dfft 2D	$T_{256} = 1.373P^{-0.883}$	0.995
3Decomp-3D	$T_{256} = 6.268P^{-0.634}$	0.962
Models for $N^3 = 512^3$		
Implementation	**Function**	R^2
Dmitruk-1D	$T_{512} = 7.163P^{-0.817}$	0.998
Ayala-2D	$T_{512} = 11.683P^{-0.926}$	0.9998
2Decomp	$T_{512} = 17.118P^{-0.89}$	0.991
Takahashi 1D	$T_{512} = 8.066P^{-0.678}$	0.986
p3dfft 2D	$T_{512} = 11.203P^{-0.84}$	0.997
3Decomp-3D	$T_{512} = 111.63P^{-0.913}$	0.9832
Models for $N^3 = 1024^3$		
Implementation	**Function**	R^2
Dmitruk-1D	$T_{1024} = 64.791P^{-0.843}$	0.996
Ayala-2D	$T_{1024} = 77.528P^{-0.812}$	0.995
2Decomp	$T_{1024} = 164.24P^{-0.907}$	0.997
Takahashi 1D	$T_{1024} = 53.839P^{-0.667}$	0.995
p3dfft 2D	$T_{1024} = 107.83P^{-0.853}$	0.997
3Decomp-3D	$T_{1024} = 2200.8P^{-1.074}$	0.998

Table 2. Power-function models for several FFT3D implementations

The experimental information obtained here supports our hypothesis: A simplified performance model for FFT3D is enough to make sufficiently accurate predictions when those predictions pertain normal operating conditions. Our model has the advantage of being significantly simpler while still being useful.

5 Summary and Conclusions

We demonstrated that a simplified performance model is enough to capture the behavior of distributed-memory implementations of FFT3D in the particular case of large-scale systems. We have targeted our efforts at predicting the performance of implementations under their typical operating conditions. That is, when enough parallelism is available and the number of processors used is reasonably large.

Preliminary results showed that the performance of FFT3D implementations followed a power-function trend. Our full set of experiments, conducted on NCAR's Bluefire supercomputer has shown that the use of a power function model is adequate to represent the performance of FFT3D implementations on traditional distributed-memory machines.

The R^2 indicator, which provides a quantitative way to measure the fidelity of an approximation, shows, in all cases, a high degree of accuracy for our method. In most cases, the R^2 indicator was greater than 0.99, further validating the effectiveness of our approach.

An important feature of the power function model is that obtaining it requires very little experimentation. The constants of the model can be obtained with as little as two measurements, while still producing useful predictions.

Our current model has been designed to produce predictions related to the execution of programs over data of a particular size. To study data of a different size, experiments must be run to determine the adjusted constants. Our future work will focus on extending our model to be a function of problem size in addition to the number of available processors.

6 Acknowledgments

This research was made possible by the generous support of the NSF through grants CCF-0833122, CCF-0925863, CCF-0937907, CNS-0720531, and OCI-0904534.

References

1. J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. pp. 297–301, 1965. [Online]. Available: http://www.jstor.org/stable/2003354
2. O. Ayala, W. W. Grabowski, and L.-P. Wang, "A hybrid approach for simulating turbulent collisions of hydrodynamically-interacting particles," *J. Comput. Phys.*, vol. 225, no. 1, pp. 51–73, Jul. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.jcp.2006.11.016
3. K. Laasonen, A. Pasquarello, R. Car, C. Lee, and D. Vanderbilt, "Car-parrinello molecular dynamics with vanderbilt ultrasoft pseudopotentials," *Physical Review B*, vol. 47, no. 16, p. 10142, 1993.
4. E. J. Bylaska, M. Valiev, R. Kawai, and J. H. Weare, "Parallel implementation of the projector augmented plane wave method for charged systems," *Computer Physics Communications*, vol. 143, no. 1, pp. 11 – 28, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010465501004131
5. H. Calandra, F. Bothorel, and P. Vezolle, "A massively parallel implementation of the common azimuth pre-stack depth migration," *IBM J. Res. Dev.*, vol. 52, no. 1/2, pp. 83–91, Jan. 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1375990.1375998
6. S. Stellmach and U. Hansen, "An efficient spectral method for the simulation of dynamos in Cartesian geometry and its implementation on massively parallel computers," *Geochemistry, Geophysics, Geosystems*, vol. 9, p. 5003, May 2008.
7. L. Wang, O. Ayala, H. Parishani, W. Grabowski, A. Wyszogrodzki, Z. Piotrowski, G. Gao, C. Kambhamettu, X. Li, L. Rossi *et al.*, "Towards an integrated multiscale simulation of turbulent clouds on petascale computers," in *Journal of Physics: Conference Series*, vol. 318. IOP Publishing, 2011, p. 072021.

8. P. Dmitruk, L.-P. Wang, W. Matthaeus, R. Zhang, and D. Seckel, "Scalable parallel fft for spectral simulations on a beowulf cluster," *Parallel Computing*, vol. 27, no. 14, pp. 1921 – 1936, 2001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016781910100120X
9. N. Li and S. Laizet, "2decompfft a highly scalable 2d decomposition library and fft interface," *Cray User Group 2010*, 2010.
10. D. Pekurovsky, "Ultrascalable fourier transfroms in three dimensions," in *Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery*, ser. TG '11. New York, NY, USA: ACM, 2011, pp. 9:1–9:2. [Online]. Available: http://doi.acm.org/10.1145/2016741.2016751
11. O. Ayala and L.-P. Wang, "Parallel implementation and scalability analysis of 3d fast fourier transform using 2d domain decomposition," *Parallel Computing (Submitted). Under review.*, 2012.
12. M. Frigo and S. Johnson, "Fftw: An adaptive software architecture for the fft," in *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, vol. 3. IEEE, 1998, pp. 1381–1384.
13. *Bluefire User Guide*. [Online]. Available: http://www2.cisl.ucar.edu/docs/bluefire-user-guide